Table of Contents

GETTING STARTED WITH EVERGREEN DEVELOPMENT

Blah Blah Blah

MAKING CONTACT

While documents have their place, there's generally no substitute for talking to existing community members - whether you're working through a tough piece of code, or putting together a patch, or just getting your development environment up and running - and you'll find that our development community tries to support newcomers like you. If you have questions, the #evergreen IRC channel on Freenode is the best place to start. You can also use the Evergreen development mailing list (open-ils-dev) if you prefer.

CORE COMMITTERS:

Name	IRC Nick	Email address	Affiliation
Galen Charlton	gmcharlt	gmc@esilibrary.com	Equinox Software
Bill Erickson	berick	erickson@esilibrary.com	Equinox Software
Jason Etheridge	phasefx	jason@esilibrary.com	Equinox Software
Lebbeous Fogle-Weekley	senator	lebbeous@esilibrary.com	Equinox Software
Mike Rylander	miker	miker@esilibrary.com	Equinox Software
Dan Scott	dbs	dan@coffeecode.net	Laurentian University / CoffeeCode Consulting
Dan Wells	dbwells	dbw2@calvin.edu	Calvin College

PRIMARY METHODS OF COMMUNICATION BETWEEN DEVELOPERS

Developers participate actively in the Development Discussion List and the IRC channel.

- Evergreen Development Discussion List: http://libmail.georgialibraries.org/mailman/listinfo/open-ils-dev
- Freenode IRC on channel #Evergreen

Many developers also participate in the General Discussion List (where users tend to hang-out).

- Evergreen General Discussion List: http://libmail.georgialibraries.org/mailman/listinfo/open-ils-general
- More about lists here: http://evergreen-ils.org/listserv.php#general

TECHNOLOGIES

Evergreen depends on the following technologies Perl, C, JavaScript, XML, XPath, XSLT, XMPP, OpenSRF, Apache, mod_perl, and PostgreSQL.

Evergreen is an OpenSRF application, so in principle you can write an OpenSRF binding for any language you want. However, in practice, Evergreen makes use of just a few languages, and for the sake of maintaining your code and understanding it, please consider sticking with one of the following.

- Perl: business logic, vast majority of OpenSRF methods used by Evergreen are written in Perl 5, not Perl 6.

- JavaScript: front end, important for botht he staff client and the public catalog (less now that Template Toolkit is being used in version 2.1)
- PL/pgSQL and PL/Perl: search code and some business logic
- HTML and CSS: public catalog and staff client
- C: core OpenSRF code; can be considered for middle-layer OpenSRF services if speed is important enough
- XUL: staff client layout

For information about OpenSRF, see Easing Gently into OpenSRF, Part 1 available at http://journal.code4lib.org/articles/3284

You should also work through this tutorial that focuses on getting up and developing with Evergreen.

Downloads and Virtual Images

The developer virtual image (OpenSRF and Evergreen trunk as of 2011-03-19 on Debian Squeeze) includes PostgreSQL 9.0 and Python support and is intended to serve as a quickstart development environment as well as a sneak peek at the latest features in Evergreen. See the README for more information.

Downloads and other virtual images area available here: http://evergreen-ils.org/downloads.php

Repositories

- Version Control Repository for Evergreen Project: http://git.evergreen-ils.org/
- List of other source code repositories: http://evergreen-ils.org/dokuwiki/doku.php?id=dev:code_repositories

Procedures and Conventions for Contributing Code to the Evergreen Project

One of the most visible ways to contribute is through actual code. All contributions and contributors will arrive at Evergreen from different backgrounds and will carry with them some legacy of their origin. Once reaching Evergreen, though, some certainties can be guaranteed:

1. Each contribution will be judged individually on its technical merits as long as the contributor has followed these guidelines
2. All contributions that are to be considered must follow project guidelines and community customs

It is expected that the person proposing a change or addition will be the one developing the patch, or in the case of a very large addition or change, leading the implementation effort of all those interested in helping code the feature. If you find an open proposal that you would like to work on with the original proposer then you should feel free to ask about collaborating.

Small Additions and Changes

- Spot some trouble, report a bug in the Evergreen bug database, and possibly propose a solution. Bug fixes and minor adjustments to code are very important to the stability and longevity of the Evergreen project, and are treated as full contributions. As such, you will need to write up a basic functional description of the suspected issue for the community to discuss as a whole. If the change is truly trivial, a core member may simply fix the issue immediately. In this case you will be listed as a bug reporter in the next release of Evergreen.

- Wait for possible feedback. You can generally expect to hear back about a minor contribution proposal within a day or two. You may just get a simple "nod" from another developer on the list, in which case it's fine to proceed to the next step. Pay attention to all replies, and proceed as soon as you get the go-ahead.
- Build and submit a patch. Put together a patch that implements the fix and post it to the open-ils-dev mailing list. You will be informed of its status by email as soon as a core team member has the chance to look at it. For patch building guidelines, please see Submitting a Patch.
- Accept and incorporate feedback. If you receive any feedback from your patch posting, please incorporate it and reply to the feedback thread, or ask for clarification if you are unsure of anything. This may seem circuitous, but the next time any other community member submits a patch they may have learned from the public feedback you received and the updated patch you sent. In any case, this is expected to be rare for minor patches.
- It's in! After your code has been committed you will be informed by email and you will appear as a contributor in the release notes for the next version of Evergreen.

Large additions and changes

- Propose a design. Because Evergreen has not only a legacy of its own but a basic road map for future implementation, any new potential features need to be weighed against the overall strategic plan and internal architecture. You will need to write up a full functional and basic implementation proposal for the community to discuss as a whole. More information about what is expected in such a proposal for discussion can be found in the section entitled How to Propose Features.
- Wait for and expect feedback. It is highly unlikely that you are the only one interested in a particular feature, and time must be given for others to comment on the basic design. How much time depends on the breadth and depth of the change or addition proposed, but no less than a few days (more, if it's over a weekend, holiday or the relevant core team members are unavailable) will suffice. Expect both positive and negative technical feedback, as there will likely be competing and parallel ideas for any non-trivial change. Discussion should surround the proposal and suggestions for improvement will be made at this time. This is where the big-picture things are hashed out.
- Lather, rinse, repeat. It is expected that, following technical feedback from the community, and particularly from existing developers, you will incorporate relevant consensus building feedback into your design. Once there is positive consensus (as opposed to "acceptance by inaction"), proceed to the next step.
- Write the code. Having come to a general consensus with the community, create some working code against your test installation. Ideally, do your work in a publicly visible repository, such as one created for you on

the Evergreen community Git server repository, a Bazaar branch, or a git repository on Gitorious or similar public repository. Working in a public repository makes it possible for others to collaborate with you.

- Submit your code to the project. As a community project, the community decides what code goes into the official releases of the software on the Evergreen downloads page. This effort is lead by the core development team, those with git commit privileges, and can be informed by anyone in the community who wants to comment. Code contributions are reviewed based on the style of the new code in relation to surrounding, pre-existing code (where applicable), equality with the proposal that met consensus, and overall quality. The process for submitting code is covered in the following section, **Submitting code to the project**.
- Accept and incorporate feedback. Contributed code, having been reviewed, may require some adjustment, and further discussion will often happen as more developers try out your code. In such cases, minor adjustments may be made by the committing core team member without further discussion.
- It's in! After your code has been committed you will be informed by email and you will appear as a contributor in the release notes for the next version of Evergreen.

External contributions

- Identify an ancillary project that you would like to integrate with, or create for the support of, Evergreen. Evergreen proper can't (and shouldn't) contain everything useful to the project or the users. If you see a need that can be fulfilled by an outside resource, but would affect the overall project (such as installation or migration projects, or external activities that point to the Evergreen project) then it serves everyone, the community and the contributor, well to get buy-in and input from the Evergreen community. You're likely to find a great deal of help and enthusiasm, as well.
- Submit a basic proposal. Create a proposal following the guidelines in listed in the section How to Propose Features to lay out a plan.
- Wait for possible feedback. You can generally expect to hear back about an external contribution proposal within a day or two. Discussion is important here, though probably won't be as extensive as a non-trivial feature proposal. Wait until consensus is reached by discussion participants, and strongly consider any suggestions and feedback they provide.
- Go forth! (But keep us up to date) Build or integrate your solution after considering all feedback. Remember to announce the project's completion (or parts thereof) on both the open-ils-dev and open-ils-general mailing lists, so that any other community members who would like to can help you continue to improve the external component.

How to Propose Features

Proposing a new non-trivial addition to the Evergreen project is fairly easy. After making sure that someone else hasn't claimed the feature by asking on the Evergreen development mailing list, collect the information required for the community to discuss the addition completely and thoroughly. The minimum requirements for this information are:

- Basic overview of the feature or change
- An explanation of the existing code – it's structure and purpose – if applicable
- An explanation of why this code needs to change, if applicable
- A full explanation of the new or replacement feature including implementation plans
- Analysis of what this change will effect in the existing code base
- Analysis of what ways, if any, this addition or change can be leveraged for future planned development

An emerging method of proposing features is to:

1. collect the information on a page on the Evergreen wiki in the dev:proposal namespace
2. add a Blueprint entry with the basic overview and a link to the corresponding wiki page
3. send the basic overview and links to the Blueprint and wiki page to the Evergreen development mailing list with a subject line beginning with Feature Proposal

For more on the procedures and conventions for contributing code to the Evergreen Project, see http://evergreen-ils.org/dokuwiki/doku.php?id=contributing.

Guidelines for Git contributors are available here: http://evergreen-ils.org/dokuwiki/doku.php?id=dev:git#guidelines_for_contributors

Development Projects

From Evergreen Libraries

These are projects that current Evergreen libraries have requested. In each case, money is available for these developments.

Make Statistical Category a required and restricted field

In the Patron Registration, libraries can designate users in a statistical category of their choosing. However, this field is not required therefore it can be bypassed during the patron registration process. In order for it to provide reliable statistical data, it must be required. Also, individual libraries can now add new entries to this table creating additional confusion. These entries should be settable by the Evergreen Admin only (or allow the Evergreen Admin to restrict them). Also, a default setting should be settable on a library by library basis.

Use case: Bibliomation uses the statistical category table to define “residency.” They would like to set the allowable fields that will apply and restrict these settings from being changed by individual libraries. In addition, the default should be associated with the local jurisdiction of the registrant.

Sponsor: Bibliomation

From Google Summer of Code

These are projects that were developed as part of the Google Summer of Code. In most cases, money would have to be sought for funding these developments because it was hoped that Google funds would cover them.

Replace remote XUL in the staff client with client XUL

- **Problem**: Mozilla has announced that the ability to run XUL from servers is deprecated, and that eventually only XUL on the client will be supported. The Evergreen GUI client used by staff is built on the XULRunner framework and - you guessed it - relies heavily on remote XUL. We need to convert our remote XUL to run on the client.
- **Required skills**: JavaScript, DOM, XML
- **Level of difficulty**: Low to Moderate
- **Mentors**: Jason Etheridge

Create Android client(s) for Evergreen

- **Problem**: Android is a great smartphone and tablet operating system. So what's the problem? An open source library system deserves great open source smartphone/tablet applications, but apart from mobile browsers on Android, there is a complete lack of Android applications that interoperate with Evergreen. However, Evergreen has a (slightly outdated, but mostly functional) Java client library to build on - so a summer of Android/Evergreen development should result in one or more Android applications. These could include:

  - a library patron app - enable a user to search the catalogue, renew items, manage holds, change their address information
  - an inventory app - attach a USB barcode scanner to a tablet and scan the collection
- **Required skills**: Java
- **Level of difficulty**:
- **Bonus points**: Provide a framework which libraries can use to create a customized version of the application. Libraries should be able to specify their particulars such as library name, logo, URLs, contact info, hours of operation, location, etc) and then click "Build" to generate a customized APK.
- **Mentors**: Dan Scott

Modernize Evergreen's Web interface

- **Problem**: Evergreen was an early adopter of the Dojo Toolkit, as its JavaScript widgets provided a powerful, approachable Web-based user interface. However, we are stuck using the 1.3 branch of Dojo (released in March, 2009) due to our dependence on some deprecated features such as the original Dojo Grid. Recent versions of Dojo offer more functionality, improved usability, better browser support, and greatly enhanced accessibility. Your job is to overcome the forces of inertia and enable Evergreen to adopt Dojo 1.6.
- **Required skills**: JavaScript
- **Level of difficulty**: Low
- **Bonus points**: Use Dojo test harnesses, or some other JavaScript testing framework, to build a test bed that ensures the quality of your own work and prevents those who follow you from desecrating your heavenly creation with regressions.
- **Mentors**: Galen Charlton, Dan Scott

Create a Ruby and/or PHP client for OpenSRF and Evergreen

- **Problem**: There are Perl, C, Python, and Java clients for Evergreen today, but the lack of a PHP client means that it is harder than it should be to integrate Evergreen with popular frameworks such as Drupal and Wordpress, as well as with library-specific applications such as VuFind. Ruby has been a popular language for a number of years and first-class support for Ruby integration would open up another class of applications.
- **Required skills**: PHP or Ruby
- **Level of difficulty**: Medium
- **Bonus points**: As you create the OpenSRF and Evergreen clients, create formal specifications for OpenSRF and the pertinent parts of Evergreen such as the fieldmapper IDL.
- **Mentors**: Dan Scott

Bring sanity to the Evergreen configuration interface

- **Problem**: There are currently 139 individual settings that a library can change within one interface in the Evergreen staff client. The presentation is a simple alphabetical list, with no further means of filtering these settings. Help for these rather complex settings is limited to a single string. There is no provision for tracking changes to the system configuration over time to correlate configuration changes to problems that may be observed. This interface is daunting for system administrators and would greatly benefit from a skilled user interface designer/implementer.
- **Required skills**: JavaScript, CSS
- **Level of difficulty**: Easy to medium
- **Mentors**: Galen Charlton

Enable metadata formats other than MARC21 to be first class citizens

- **Problem**: While Evergreen is built on an advanced platform, much of its database schema, search indexing and results display, and tooling assumes that the loved-by-libraries-but-oh-so-limited MARC21 is the sole format in which metadata will be stored and maintained. To support a broader base of knowledge-based institutions, many of which use more modern metadata formats, Evergreen needs to be taught how to ingest, index, display, and edit other metadata formats without requiring them to be converted to MARC21.
- **Required skills**: SQL, XML, XSLT
- **Level of difficulty**: Medium to hard
- **Mentors**: Dan Scott, Galen Charlton

Offer management and integration of full-text search of objects within Evergreen

- **Problem**: Evergreen is currently limited to searching metadata, but our knowledge institutions are increasingly moving towards offering either born digital objects (ebooks, electronic journals, theses and dissertations) or digitized works. Consequently, a frequently asked question is whether there is a way to search full-text objects along with just metadata, and the current answer is “no”. One way to change this answer is to tightly integrate Evergreen with an existing digital repository such as DSpace or Fedora; alternately, one could add full-text digital object support directly to Evergreen.
- **Required skills**: Full-text search (Solr, Sphinx, …)
- **Level of difficulty**: High
- **Mentors**: Galen Charlton

Overhaul internationalization support in Evergreen

- **Problem**: Internationalization (i18n) support in the public interface and most of the staff client was added several years ago, but while an effort was made to standardize on GNU gettext as an intermediate format, the effort was somewhat rudimentary and many new interfaces have subsequently been added that lack i18n support. Given the increasing adoption of Evergreen outside of North America, the time is right to revisit some of the initial i18n approaches and deliver a consistent approach to i18n throughout the application, with an eye towards the addition of right-to-left support for bidirectional languages; locale-sensitive currency, time, and date support; and use of a shared system across timezones.

- **Required skills**: JavaScript, Perl
- **Level of difficulty**: Medium
- **Mentors**: Dan Scott

ADD CUSTOMIZABLE TOOLBARS TO THE EVERGREEN STAFF CLIENT

- **Problem**: The Evergreen Staff Client has a single toolbar that can be shown or not. The buttons on it are hard coded into the XUL interface, and as such it cannot be easily customized for different libraries, workstations, user accounts, and so forth. Ideally there would be multiple default toolbars for different functions (such as cataloging or circulation), and those and new ones would be customizable by workstation, library, or user.
- **Required skills**: JavaScript, DOM
- **Level of difficulty**: Low
- **Mentors**: Jason Etheridge

BUILD EASIER CONFIGURATION INTERFACE FOR INDEXING DEFINITIONS

- **Problem**: Currently Evergreen requires the use of XSLT and XPath to configure the mapping of metadata field to search indexes. This is not the friendliest of interfaces for many librarians to use. A interface that allows the expression of index definitions in terms more familiar to librarians (e.g., MARC tags, headings, and named metadata fields) would make it much easier for librarians to customize their use of Evergreen.
- **Required skills**: JavaScript, XML, XSLT, Perl
- **Level of difficulty**: Medium
- **Mentors**: Galen Charlton

BUILD A CUSTOMIZABLE DASHBOARD TO SHOW LIBRARY USAGE AND SERVER HEALTH

- **Problem**: Up-to-date information about the use of the library and the health of its servers is available in reports and logfiles, but few stakeholders have the access, expertise, and initiative to check those sources. A dashboard framework with useful starter components would make this information ambiently available with just a glance. It would be useful to be able to show the current number of loaned/overdue items; current total overdues owed; average number of renewals per circ modifier; number of circulations in/out per ***n*** minutes. For internal use, it would also be useful to show server load, free memory and disk space, etc. as recorded by nagios, MRTG, et al. http://library.brown.edu/dashboard/info/ has some great ideas.

- **Required skills**:
- **Level of difficulty**:
- **Mentors**: Dan Scott